Lecture 02: Conceptual Design

Wednesday, October 6, 2010
Nulls

- \texttt{count(category)} \neq \texttt{count(*)} \ WHY ?

- Office hours: Thursdays, 5-6pm
Announcements

• Homework 2 is posted: due October 19th
• You need to create tables, import data:
  – On SQL Server, in your own database, OR
  – On postgres (we will use it for Project 2)
• Follow Web instructions for importing data
• Read book about CREATE TABLE, INSERT, DELETE, UPDATE
Discussion

SQL Databases v. NoSQL Databases, Mike Stonebraker

• What are “No-SQL Databases”?
• What are the two main types of workloads in a database? (X and Y)
• How can one improve performance of X?
• Where does the time of a single server go?
• What are “single-record transactions”? 
Outline

• E/R diagrams

• From E/R diagrams to relations
Database Design

• Why do we need it?
  – Agree on structure of the database before deciding on a particular implementation.

• Consider issues such as:
  – What entities to model
  – How entities are related
  – What constraints exist in the domain
  – How to achieve good designs

• Several formalisms exists
  – We discuss E/R diagrams
Entity / Relationship Diagrams

Objects → entities
Classes → entity sets

Attributes:

Relationships

- first class citizens (not associated with classes)
- not necessarily binary
Keys in E/R Diagrams

- Every entity set must have a key
- May be a *multi-attribute key*:

```
Product
  prod-ID
  category
  price

Order
  prod-ID
  cust-ID
  date
```
What is a Relation?

A mathematical definition:
- if $A$, $B$ are sets, then a relation $R$ is a subset of $A \times B$

$A = \{1, 2, 3\}$, $B = \{a, b, c, d\}$,

$A \times B = \{(1,a), (1,b), \ldots, (3,d)\}$

$R = \{(1,a), (1,c), (3,b)\}$

- makes is a subset of $\text{Product} \times \text{Company}$:
Multiplicity of E/R Relations

- one-one:

- many-one

- many-many
Notation in Class v.s. the Book

In class:

Product \(\rightarrow\) makes \(\rightarrow\) Company

In the book:

Product \(\rightarrow\) makes \(\rightarrow\) Company
Multi-way Relationships

Product

Person

Purchase

date

Store
Converting Multi-way Relationships to Binary

Arrows are missing: which ones?
3. Design Principles

What’s wrong?

Product → Purchase ← Person

Country → President → Person

Moral: be faithful!
Design Principles: What’s Wrong?

Moral: pick the right kind of entities.
Design Principles: What’s Wrong?

Moral: don’t complicate life more than it already is.
From E/R Diagrams to Relational Schema

- Entity set $\rightarrow$ relation
- Relationship $\rightarrow$ relation
Entity Set to Relation

**Product**(*prod-ID*, category, price)

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Camera</td>
<td>99.99</td>
</tr>
<tr>
<td>Pokemn19</td>
<td>Toy</td>
<td>29.99</td>
</tr>
</tbody>
</table>
Create Table (SQL)

CREATE TABLE Product (  
    prod-ID CHAR(30) PRIMARY KEY,  
    category VARCHAR(20),  
    price double)
Relationships to Relations

Shipment \((prod-ID, cust-ID, name, date)\)

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>cust-ID</th>
<th>name</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>UPS</td>
<td>4/10/2010</td>
</tr>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>FEDEX</td>
<td>4/9/2010</td>
</tr>
</tbody>
</table>
CREATE TABLE Shipment(
    name CHAR(30)
    REFERENCES Shipping-Co,
    prod-ID CHAR(30),
    cust-ID VARCHAR(20),
    date DATETIME,
    PRIMARY KEY (name, prod-ID, cust-ID),
    FOREIGN KEY (prod-ID, cust-ID)
    REFERENCES Orders
)
Multi-way Relationships to Relations

Product
  prod-ID
  price

Purchase

Person
  ssn
  name

Store
  name
  address

How do we represent that in a relation?
Modeling Subclasses

Products

- Software products
- Educational products
Subclasses

Product

isa

Software Product

is a

Educational Product

name
category
price

platforms

Age Group

Dan Suciu -- CSEP544 Fall 2010
Understanding Subclasses

• Think in terms of records:
  – Product
    - field1
    - field2
  – SoftwareProduct
    - field1
    - field2
    - field3
  – EducationalProduct
    - field1
    - field2
    - field3
    - field4
    - field5
Subclasses to Relations

Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>

Software Product

<table>
<thead>
<tr>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
</tr>
</tbody>
</table>

Educational Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Age Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>todler</td>
</tr>
<tr>
<td>Toy</td>
<td>retired</td>
</tr>
</tbody>
</table>

Dan Suciu -- CSEP544 Fall 2010
Modeling UnionTypes With Subclasses

FurniturePiece

Person

Company

Say: each piece of furniture is owned either by a person, or by a company
Modeling Union Types with Subclasses

Say: each piece of furniture is owned either by a person, or by a company

Solution 1. Acceptable (What’s wrong?)
Modeling Union Types with Subclasses

Solution 2: More faithful

Diagram:
- Owner
  - isa: Person
  - isa: Company
- ownedBy: FurniturePiece

Dan Suciu -- CSEP544 Fall 2010
Constraints in E/R Diagrams

Finding constraints is part of the modeling process. Commonly used constraints:

- **Keys**: social security number uniquely identifies a person.

- **Single-value constraints**: a person can have only one father.

- **Referential integrity constraints**: if you work for a company, it must exist in the database.

- **Other constraints**: peoples’ ages are between 0 and 150.
Keys in E/R Diagrams

Underline:

- name
- category
- price

Product

Multi-attribute key v.s. Multiple keys

Not possible in E/R
Single Value Constraints
Referential Integrity Constraints

Each product made by at most one company. Some products made by no company

Each product made by exactly one company.
Other Constraints

What does this mean?
Weak Entity Sets

Entity sets are weak when their key comes from other classes to which they are related.

Notice: we encountered this when converting multiway relationships to binary relationships.
Handling Weak Entity Sets

How do we represent this with relations?
Weak Entity Sets

Weak entity set = entity where part of the key comes from another number

Convert to a relational schema (in class)
What Are the Keys of R?
Design Theory
Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = will study
- 3rd Normal Form = see book
First Normal Form (1NF)

• A database schema is in First Normal Form if all tables are flat.

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
</tr>
</tbody>
</table>

May need to add keys
Relational Schema Design

Conceptual Model:

Relational Model: plus FD’s

Normalization: eliminates *anomalies*
Data Anomalies

When a database is poorly designed we get anomalies:

**Redundancy**: data is repeated

**Updated anomalies**: need to change in several places

**Delete anomalies**: may lose data when we don’t want
Relational Schema Design

Recall set attributes (persons with several phones):

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

One person may have multiple phones, but lives in only one city

**Anomalies:**

- Redundancy = repeat data
- Update anomalies = Fred moves to “Bellevue”
- Deletion anomalies = Joe deletes his phone number: what is his city?
Relation Decomposition

Break the relation into two:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>206-555-1234</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>206-555-6543</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-2121</td>
</tr>
</tbody>
</table>

Anomalies have gone:

- No more repeated data
- Easy to move Fred to “Bellevue” (how ?)
- Easy to delete all Joe’s phone number (how ?)
Relational Schema Design (or Logical Design)

Main idea:

• Start with some relational schema
• Find out its functional dependencies
• Use them to design a better relational schema
Functional Dependencies

• A form of constraint
  – hence, part of the schema
• Finding them is part of the database design
• Also used in normalizing the relations
Functional Dependencies

Definition:

If two tuples agree on the attributes

\[ A_1, A_2, \ldots, A_n \]

then they must also agree on the attributes

\[ B_1, B_2, \ldots, B_m \]

Formally:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
When Does an FD Hold

Definition: \( A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \) holds in \( R \) if:

\[
\forall t, t' \in R, (t.A_1 = t'.A_1 \land \ldots \land t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \land \ldots \land t.B_n = t'.B_n)
\]

<table>
<thead>
<tr>
<th>R</th>
<th>A_1</th>
<th>...</th>
<th>A_m</th>
<th>B_1</th>
<th>...</th>
<th>n_m</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If \( t, t' \) agree here, then \( t, t' \) agree here.
An FD holds, or does not hold on an instance:

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

EmpID $\rightarrow$ Name, Phone, Position
Position $\rightarrow$ Phone
but not Phone $\rightarrow$ Position
## Example

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

**Position $\rightarrow$ Phone**
### Example

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

but not Phone → Position
Example

FD’s are constraints:
On some instances they hold
On others they don’t

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Does this instance satisfy all the FDs?
## Example

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
An Interesting Observation

If all these FDs are true:

\[
\begin{align*}
\text{name} &\rightarrow \text{color} \\
\text{category} &\rightarrow \text{department} \\
\text{color}, \text{category} &\rightarrow \text{price}
\end{align*}
\]

Then this FD also holds:

\[
\text{name, category} \rightarrow \text{price}
\]

Why ??
Goal: Find ALL Functional Dependencies

• Anomalies occur when certain “bad” FDs hold

• We know some of the FDs

• Need to find all FDs, then look for the bad ones
Armstrong’s Rules (1/3)

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

Is equivalent to

\[ A_1, A_2, \ldots, A_n \rightarrow B_1 \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_2 \]

\[ \ldots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]
Armstrong’s Rules (2/3)

\[
A_1, A_2, \ldots, A_n \rightarrow A_i
\]

where \( i = 1, 2, \ldots, n \)

Why?

Trivial Rule

Dan Suciu -- CSEP544 Fall 2010
Armstrong’s Rules (3/3)

Transitive Closure Rule

If \[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

and \[ B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p \]

then \[ A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p \]

Why?
Example (continued)

Start from the following FDs:

1. name \rightarrow\ color
2. category \rightarrow\ department
3. color, category \rightarrow\ price

Infer the following FDs:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category \rightarrow name</td>
<td></td>
</tr>
<tr>
<td>5. name, category \rightarrow color</td>
<td></td>
</tr>
<tr>
<td>6. name, category \rightarrow category</td>
<td></td>
</tr>
<tr>
<td>7. name, category \rightarrow color, category</td>
<td></td>
</tr>
<tr>
<td>8. name, category \rightarrow price</td>
<td></td>
</tr>
</tbody>
</table>
**Example (continued)**

**Answers:**

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category → name</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>5. name, category → color</td>
<td>Transitivity on 4, 1</td>
</tr>
<tr>
<td>6. name, category → category</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>7. name, category → color, category</td>
<td>Split/combine on 5, 6</td>
</tr>
<tr>
<td>8. name, category → price</td>
<td>Transitivity on 3, 7</td>
</tr>
</tbody>
</table>

THIS IS TOO HARD! Let’s see an easier way.
Closure of a set of Attributes

Given a set of attributes \( A_1, \ldots, A_n \)

The closure, \( \{A_1, \ldots, A_n\}^+ = \) the set of attributes \( B \)

s.t. \( A_1, \ldots, A_n \rightarrow B \)

Example:

- \( \text{name} \rightarrow \text{color} \)
- \( \text{category} \rightarrow \text{department} \)
- \( \text{color, category} \rightarrow \text{price} \)

Closures:

- \( \text{name}^+ = \{\text{name, color}\} \)
- \( \{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\} \)
- \( \text{color}^+ = \{\text{color}\} \)
Closure Algorithm

\[ X = \{A_1, \ldots, A_n\} \]

**Repeat until** \( X \) doesn’t change **do:**

- **if** \( B_1, \ldots, B_n \rightarrow C \) is a FD **and**
  - \( B_1, \ldots, B_n \) are all in \( X \)
- **then** add \( C \) to \( X \).

\[ \{\text{name, category}\}^+ = \{ \} \]

**Hence:** \( \text{name, category} \rightarrow \text{color, department, price} \)

Example:

- name \( \rightarrow \) color
- category \( \rightarrow \) department
- color, category \( \rightarrow \) price
Example

In class:

\[ R(A,B,C,D,E,F) \]

\[
\begin{array}{|c|}
\hline
A, B & \rightarrow & C \\
A, D & \rightarrow & E \\
B & \rightarrow & D \\
A, F & \rightarrow & B \\
\hline
\end{array}
\]

Compute \( \{A,B\}^+ \):

\[ X = \{A, B, \} \]

Compute \( \{A, F\}^+ \):

\[ X = \{A, F, \} \]
Why Do We Need Closure

• With closure we can find all FD’s easily

• To check if $X \rightarrow A$
  – Compute $X^+$
  – Check if $A \in X^+$
Using Closure to Infer ALL FDs

Example:

A, B \rightarrow C
A, D \rightarrow B
B \rightarrow D

Step 1: Compute $X^+$, for every $X$:

$A^+ = A$, $B^+ = BD$, $C^+ = C$, $D^+ = D$

$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$,

$BC^+ = BCD$, $BD^+ = BD$, $CD^+ = CD$

$ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute– why ?)

$BCD^+ = BCD$, $ABCD^+ = ABCD$

Step 2: Enumerate all FD’s $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$
Another Example

- Enrollment(student, major, course, room, time)
  - student $\rightarrow$ major
  - major, course $\rightarrow$ room
  - course $\rightarrow$ time

What else can we infer? [in class, or at home]
Keys

• **A superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute $B$, we have $A_1, ..., A_n \rightarrow B$

• **A key** is a minimal superkey
  – I.e. set of attributes which is a superkey and for which no subset is a superkey
Computing (Super)Keys

- Compute $X^+$ for all sets $X$
- If $X^+ = \text{all attributes}$, then $X$ is a key
- List only the minimal $X$’s
Example

Product(name, price, category, color)

<table>
<thead>
<tr>
<th>name, category</th>
<th>→</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>→</td>
<td>color</td>
</tr>
</tbody>
</table>

What is the key?
Example

Product(name, price, category, color)

(name, category) \rightarrow price
category \rightarrow color

What is the key?

(name, category) + = name, category, price, color

Hence (name, category) is a key
Examples of Keys

Enrollment(student, address, course, room, time)

<table>
<thead>
<tr>
<th>student</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>room, time</td>
<td>course</td>
</tr>
<tr>
<td>student, course</td>
<td>room, time</td>
</tr>
</tbody>
</table>

(find keys at home)
Eliminating Anomalies

Main idea:

• $X \rightarrow A$ is OK if $X$ is a (super)key

• $X \rightarrow A$ is not OK otherwise
## Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN → Name, City

What the key?}
Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN → Name, City

What the key? Hence SSN → Name, City

{SSN, PhoneNumber} is a “bad” dependency
Key or Keys?

Can we have more than one key?

Given \( R(A,B,C) \) define FD’s s.t. there are two or more keys
Key or Keys?

Can we have more than one key?

Given R(A,B,C) define FD’s s.t. there are two or more keys

\[
\begin{align*}
AB & \rightarrow C \\
BC & \rightarrow A
\end{align*}
\]

or

\[
\begin{align*}
A & \rightarrow BC \\
B & \rightarrow AC
\end{align*}
\]

what are the keys here?

Can you design FDs such that there are three keys?
Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation $R$ is in BCNF if:

- If $A_1, ..., A_n \rightarrow B$ is a non-trivial dependency
- in $R$, then $\{A_1, ..., A_n\}$ is a superkey for $R$

In other words: there are no “bad” FDs

Equivalently:

$$\forall X, \text{ either } (X^+ = X) \text{ or } (X^+ = \text{all attributes})$$
BCNF Decomposition Algorithm

repeat
  choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates BNCF
  split $R$ into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, [\text{others}])$
  continue with both $R_1$ and $R_2$
until no more violations

In practice, we have a better algorithm (coming up)
Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN $\rightarrow$ Name, City

What the key?

\[ \{ \text{SSN, PhoneNumber} \} \quad \text{use SSN $\rightarrow$ Name, City to split} \]
Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN $\rightarrow$ Name, City

Let’s check anomalies:
Redundancy ?
Update ?
Delete ?
Example Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN $\rightarrow$ name, age
age $\rightarrow$ hairColor

Decompose in BCNF (in class):
BCNF Decomposition Algorithm

BCNF_Decompose(R)

find X s.t.: X ≠ X⁺ ≠ [all attributes]

if (not found) then “R is in BCNF”

let Y = X⁺ - X
let Z = [all attributes] - X⁺

decompose R into R1(X ∪ Y) and R2(X ∪ Z)
continue to decompose recursively R1 and R2
Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN $\rightarrow$ name, age
age $\rightarrow$ hairColor

Find $X$ s.t.: $X \neq X^+ \neq$ [all attributes]
Find X s.t.: X ≠ X⁺ ≠ [all attributes]

Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)
  SSN → name, age
  age → hairColor

Iteration 1: Person: SSN⁺ = SSN, name, age, hairColor
Decompose into: P(SSN, name, age, hairColor)
                 Phone(SSN, phoneNumber)

Iteration 2: P: age⁺ = age, hairColor
Decompose: People(SSN, name, age)
            Hair(age, hairColor)
            Phone(SSN, phoneNumber)
Example

\[ R(A, B, C, D) \]

\[ A^+ = ABC \neq ABCD \]
What are the keys?

$R(A,B,C,D)$

$A^+ = ABC \neq ABCD$

$R_1(A,B,C)$

$B^+ = BC \neq ABC$

$R_{11}(B,C)$

$R_{12}(A,B)$

What happens if in $R$ we first pick $B^+$? Or $AB^+$?

A $\rightarrow$ B

B $\rightarrow$ C

Dan Suciu -- CSEP544 Fall 2010
Decompositions in General

\[ R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p) \]

\[ R_1(A_1, ..., A_n, B_1, ..., B_m) \]

\[ R_2(A_1, ..., A_n, C_1, ..., C_p) \]

\[ R_1 = \text{projection of } R \text{ on } A_1, ..., A_n, B_1, ..., B_m \]

\[ R_2 = \text{projection of } R \text{ on } A_1, ..., A_n, C_1, ..., C_p \]
Theory of Decomposition

Sometimes it is correct:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

Lossless decomposition
Incorrect Decomposition

Sometimes it is not:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

What’s incorrect??

Lossy decomposition
Decompositions in General

\[ R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p) \]

If \( A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \)

Then the decomposition is lossless

Note: don’t need \( A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p \)

BCNF decomposition is always lossless. WHY?